Relational Algebra (Part 1)

Instructor: David Martin

Reference:
A First Course in Database Systems, 3rd edition, Chapter 2.4 – 2.6

What is a Data Model?

- A *data model* is a mathematical formalism that consists of three parts:
  1. A notation for describing and representing data (<u>structure</u> of the data)
  2. A set of <u>operations</u> for manipulating data.
  3. A means of describing <u>constraints</u> on the data.

- What is the associated query language for the relational data model?

Two Formal Query Languages

- Codd proposed two different query languages for the relational data model.
  - Relational Algebra
    - Queries are expressed as a sequence of operations on relations.
    - “Procedural” language.
  - Relational Calculus
    - Queries are expressed as formulas of first-order logic.
    - “Declarative” language.
- **Codd’s Theorem**: The Relational Algebra query language has the same *expressive power* as the Relational Calculus query language.

Procedural vs. Declarative Languages

(Out-of-Scope for Exams)

- **Procedural program**
  - The program is specified as a sequence of operations to obtain the desired outcome. I.e., *how* the outcome is to be obtained
  - More explicit about procedures
  - E.g., Java, C, …
- **Declarative program**
  - The program specifies *what* is the expected outcome, and knowledge the system needs to obtain the outcome
  - More explicit about knowledge/meaning
  - E.g., Scheme, Ocaml, Prolog …

Procedural vs. Declarative Languages (2)

(Out-of-Scope for Exams)

- Is SQL a procedural or a declarative language?
  - SQL is usually described as declarative, but it's not fully declarative
  - Relational database systems usually try to understand meaning of query, regardless of how query is expressed

Relational Algebra

- Relational Algebra: a query language for manipulating data in the relational data model.
  - Not used directly as a query language in commercial systems

- Internally, Relational Database Systems transform SQL queries into trees/graphs that are similar to relational algebra expressions.
  - Query analysis, transformation and optimization are performed based on these relational algebra expression-like representations.
  - Relational Databases use multi-sets/bags, but Relational Algebra is based on <u>sets</u>.
    - There are variations of Relational Algebra that permit duplicates, but we won't cover those.

Why Study Relational Algebra?

- Reinforces the fundamental ideas / operations embodied in SQL
- Shows a bit of what query planners and optimizers do, and what it's based upon
- It's a significant achievement of theoretical computer science
- Provides a lovely example of how computer science theory supports applications in products
- If you ever work on implementing (certain parts of) a DBMS, you'll need to know it
- If you ever work on database theory, you'll need to know it

Composition

- Each Relational Algebra operator is either a unary or a binary operator.
- A complex Relational Algebra expression is built up from basic ones by composing simpler expressions.
- This is similar to SQL queries and views.

Relation Algebra Operators

- Queries in relational algebra are composed using basic operations or functions.
  - Selection (σ)
  - Projection (π)
  - Set-theoretic operations:
    - Union (⋃)
    - Set-difference (-)
    - Cross-product (x)
    - Intersection (⋂)
  - Renaming (ρ)
  - Natural Join (⋈), Theta-Join ($\bowtie_\theta$)
  - Division (/ or ÷)

Relational Algebra Operators

- Codd proved that the relational algebra operators (σ , π , x , U , -, ρ) are independent of each other. That is, you can't define any of these operators using the others.

- However, there are other important operators that can be expressed using (σ , π , x , U , -)
  - Natural Join, Theta Join, Semi-Join
  - Set Intersection
  - Division (We will skip this; if slides are present you may ignore them)
  - Outer Join (out of scope for exams, but we may touch on this later)

Selection: $\sigma_{condition}(R)$

- Unary operation
  - Input: Relation with schema $R(A_1, \ldots, A_n)$
  - Output: Relation with attributes $A_1, \ldots, A_n$
  - Meaning: Takes a relation R and extracts only the rows from R that satisfy the *condition*
  - Condition is a logical conjunction (using AND, OR, NOT) of expressions of the form:

    <expr> <op> <expr>

    where <expr> is an attribute name, a constant, a string, and op is one of (=, ≤, ≥, <, >, <>)
    - E.g., “age > 20 OR height < 6”,
    - “name LIKE “Anne%” AND salary > 200000”
    - “NOT (age > 20 AND salary < 100000)”

Example of σ

- $\sigma_{\text{rating} > 6}$ (Hotels)

Hotels

name	address	rating	capacity
~~Windsor~~	~~54^{th} ave~~	~~6.0~~	~~135~~
Astoria	5^{th} ave	8.0	231
BestInn	45^{th} st	6.7	28
~~ELodge~~	~~39 W st~~	~~5.6~~	~~45~~
~~ELodge~~	~~2nd E st~~	~~6.0~~	~~40~~

name	address	rating	capacity
Astoria	5^{th} ave	8.0	231
BestInn	45^{th} st	6.7	28

Example of σ with AND in Condition

- $\sigma_{\text{rating > 6 AND capacity > 50}}$ (Hotel)

name	address	rating	capacity
Windsor	54th ave	6.0	135
Astoria	5th ave	8.0	231
BestInn	45th st	6.7	28
ELodge	39 W st	5.6	45
ELodge	2nd E st	6.0	40

name	address	rating	capacity
Astoria	5th ave	8.0	231

- Is $\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C1 \text{ AND } C2}(R)$?
- Prove or give a counter-example.

- Is $\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C2}(\sigma_{C1}(R))$?
- Prove or give a counter-example.

(this box is out-of-scope for exams)

Projection: $\pi_{\textit{<attribute list>}}(R)$

- Unary operation
  - Input : Relation with schema $R(A_1, \ldots, A_n)$
  - Output: Relation with attributes in *attribute list,* which must be attributes of R
  - Meaning: For every tuple in relation R, output only the attributes appearing in *attribute list*
- May be duplicates; for Codd's Relational Algebra, duplicates are always eliminated (set-oriented semantics)
  - Reminder: For relational database, duplicates matter.

Example of π

- $\pi_{\text{name, address}}$ (Hotels)

name	address
Windsor	54^{th} ave
Astoria	5^{th} ave
BestInn	45^{th} st
ELodge	39 W st
ELodge	2nd E st

- Suppose that name and address form the key of the Hotels relation. Is the cardinality of the output relation the same as the cardinality of Hotels? Why?

Example of π

- π_{name} (Hotel)

name
Windsor
Astoria
BestInn
ELodge

- Note that there are no duplicates.

Set Union: R U S

- Binary operator
  - Input: Two relations R and S which must be union-compatible
    - They have the same arity, i.e., the same number of columns.
    - For every column i, the i'th column of R has the same type as the i'th column of S.
    - Note that field names are not used in defining union-compatibility.
      - We can think of relations R and S as being union-compatible if they are sets of records having the same record type.
  - Output: Relation that has the same structure as R (and same as S).
  - Meaning: The output consists of the set of all tuples in either R or S (or both)

Example of ∪

Dell_Desktops ∪ IBM_Desktops

Dell_Desktops

Harddisk	Speed	OS
20G	500Mhz	Windows
30G	1.0Ghz	Windows
20G	750Mhz	Linux

IBM_Desktops

Harddisk	Speed	OS
30G	1.2Ghz	Windows
20G	500Mhz	Windows

All tuples in R occurs in R ∪ S.
All tuples in S occurs in R ∪ S.
R ∪ S contains tuples that either occur in R or S (or both).

Harddisk	Speed	OS
20G	500Mhz	Windows
30G	1.0Ghz	Windows
20G	750Mhz	Linux
30G	1.2Ghz	Windows

Properties of U

Dell_Desktops U IBM_Desktops

RUS = SUR (commutativity)
(RUS)UT = RU(SUT) (associativity)
(this box is out-of-scope)

Dell_Desktops

Harddisk	Speed	OS
20G	500Mhz	Windows
30G	1.0Ghz	Windows
20G	750Mhz	Linux

IBM_Desktops

Harddisk	Speed	OS
30G	1.2Ghz	Windows
20G	500Mhz	Windows

Harddisk	Speed	OS
20G	500Mhz	Windows
30G	1.0Ghz	Windows
20G	750Mhz	Linux
30G	1.2Ghz	Windows

Set Difference: R - S

- Binary operator.
  - Input: Two relations R and S which must be union-compatible
  - Output: Relation with the same type as R (or same type as S)

  - Meaning: Output consists of all tuples in R but <u>not</u> in S

Example of -

- Dell_Desktops - IBM_Desktops

Dell_Desktops

Harddisk	Speed	OS
20G	500Mhz	Windows
30G	1.0Ghz	Windows
20G	750Mhz	Linux

IBM_Desktops

Harddisk	Speed	OS
30G	1.2Ghz	Windows
20G	500Mhz	Windows

Dell_Desktops – IBM_Desktops

Harddisk	Speed	OS
30G	1.0Ghz	Windows
20G	750Mhz	Linux

Properties of -

- IBM_Desktops – Dell_Desktops

Dell_Desktops

Harddisk	Speed	OS
20G	500Mhz	Windows
30G	1.0Ghz	Windows
20G	750Mhz	Linux

IBM_Desktops

Harddisk	Speed	OS
30G	1.2Ghz	Windows
20G	500Mhz	Windows

IBM_Desktops – Dell_Desktops

Harddisk	Speed	OS
30G	1.2Ghz	Windows

Is it commutative?
Is it associative?

(this box is out-of-scope for exams)

Product: R x S

- Binary operator
  - Input: Two relations R and S, where R has relation schema $R(A_1, \ldots, A_m)$ and S has relation schema $S(B_1, \ldots, B_n)$.
  - Output: Relation of arity m+n

  - Meaning:

    $R \times S = \{ (a_1, \ldots, a_m, b_1, \ldots, b_n) \mid (a_1, \ldots, a_m) \in R \text{ and } (b_1, \ldots, b_n) \in S) \}$.

    - Read "|" as "such that"
    - Read "∈" as "belongs to"

Example and Properties of Product

R

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

S

D	E
d_1	e_1
d_2	e_2
d_3	e_3

R x S

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_1	d_2	e_2
a_1	b_1	c_1	d_3	e_3
a_2	b_2	c_2	d_1	e_1
a_2	b_2	c_2	d_2	e_2
a_2	b_2	c_2	d_3	e_3

- Is it commutative?
- Is it associative?
- Is it distributive across ∪? That is, does Rx(S∪T) = (RxS) ∪ (RxT) ?

(this box is out-of-scope for exams)

Product and Common Attributes

- What happens when we compute the Product of R and S if R and S contain common attributes, e.g., for R(A,B,C) and S(A,E)?

R.A	B	C	S.A	E
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_1	d_2	e_2
a_1	b_1	c_1	d_3	e_3
a_2	b_2	c_2	d_1	e_1
a_2	b_2	c_2	d_2	e_2
a_2	b_2	c_2	d_3	e_3

Renaming: $\rho_{S(A1, \ldots, An)}$ (R)

- To specify the attributes of a relational expression.
- Input: a relation, a relation symbol R, and a set of attributes {B1, … ,Bn}
- Output: the same relation with name S and attributes A1, …, An.

- Meaning: rename relation R to S with attributes A1, …, An.

- Example: $\rho_{\text{BeersInfo(beer,maker)}}$ Beers(name, manuf)

Renaming Example

R

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

S

C	D
d_1	e_1
d_2	e_2
d_3	e_3

$R \times \rho_{T(X,D)} S$

A	B	C	X	D
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_1	d_2	e_2
a_1	b_1	c_1	d_3	e_3
a_2	b_2	c_2	d_1	e_1
a_2	b_2	c_2	d_2	e_2
a_2	b_2	c_2	d_3	e_3

Derived Operators

- So far, we have learned:
  - Selection
  - Projection
  - Product
  - Union
  - Difference
  - Renaming
- Some other operators can be derived by composing the operators we have learned so far:
  - Set Intersection
  - Natural Join, Theta-Join, Semi-Join
  - Division (won’t spend time on this)
  - Outer Join (to be discussed when we get to OLAP)

Set Intersection: R∩S

Find all desktops sold by both Dell and IBM.

Dell_desktops ∩ IBM_desktops

Dell_Desktops

Harddisk	Speed	OS
20G	500Mhz	Windows
30G	1.0Ghz	Windows
20G	750Mhz	Linux

IBM_Desktops

Harddisk	Speed	OS
30G	1.2Ghz	Windows
20G	500Mhz	Windows

Harddisk	Speed	OS
20G	500Mhz	Windows

Intersection

- How would you write Dell_desktops ∩ HP_desktops in SQL?

```
SELECT *
FROM Dell_desktops
INTERSECT
SELECT *
FROM HP_desktops;
```

- Intersection is a <u>Derived Operator</u> in Relational Algebra:

$$
\begin{aligned}
R \cap S &= R - (R - S) \\
&= S - (S - R)
\end{aligned}
$$

Theta-Join: $R \bowtie_\theta S$

- Binary operator
  - Input: $R(A_1, \ldots, A_m)$, $S(B_1, \ldots, B_n)$
  - Output: Relation consisting of all attributes $A_1, \ldots, A_m$ and all attributes $B_1, \ldots, B_n$. Identical attributes in R and S are disambiguated with the relation names.

  - Meaning of $\sigma_\theta(R \times S)$: The θ-Join outputs those tuples from R x S that satisfy the condition θ.
    - Compute R x S, then keep only those tuples in R x S that satisfy θ.
    - Equivalent to writing $\sigma_\theta(R \times S)$

- If θ always evaluates to true, then $R \bowtie_\Theta S = \sigma_\theta(R \times S) = R \times S$.

Example of Theta-Join

Enrollment(esid, ecid, grade)

Course(cid, cname, instructor-name)

Write a Theta-Join on the board where ecid in Enrollment equals cid in Course.

- Joins involving equality predicates (usually just called Joins or Equi-Joins) are very common in database; other joins are less common.
  - Enrollment $\bowtie_{\theta}$ Course, where θ could be:
    "Enrollment.ecid = Course.cid"

- Could write <u>any</u> condition involving attributes of Enrollment and Course as θ, just as with σ.

Natural Join: R⋈S

- Often a query over two relations can be formulated using Natural Join.
- Binary operator:
  - Input: Two relations R and S where { A_1, …, A_k } is the set of common attributes (column names) between R and S.
  - Output: A relation where its attributes are attr(R) U attr(S). In other words, the attributes consists of the attributes in R x S <u>without</u> duplication of the common attributes { A_1, …, A_k }

- Meaning:

  $R \bowtie S = \pi_{(attr(R) \cup attr(S))} (\sigma_{R.A1=S.A1 \text{ AND } R.A2 = S.A2 \text{ AND } \ldots \text{ AND } R.Ak=S.Ak} (R \times S))$

1. Compute R x S
2. Keep only those tuples in R x S satisfying:
   R.A1=S.A1 AND R.A2 = S.A2 AND … AND R.Ak=S.Ak
3. Output is projection on the set of attributes in R U S

Natural Join Example

Enrollment x Course

Enrollment

studentID	courseID	grade
112	CMPS101	C
327	CMPS101	A
835	BINF223	B

Course

courseID	description	department
CMPS101	Algo.	CS
BINF223	Intro. to bio.	Biology

Natural Join Example

$$R \bowtie S = \pi_{(attr(R) \cup attr(S))} (\sigma_{R.A1=S.A1 \text{ AND } R.A2 = S.A2 \text{ AND } \ldots \text{ AND } R.Ak=S.Ak} (R \times S))$$

Step 1: R x S

Enrollment	Course

studentID	courseID	grade	courseID	description	department
112	CMPS101	C	CMPS101	Algo.	CS
112	CMPS101	C	BINF223	Intro. to bio.	Biology
327	CMPS101	A	CMPS101	Algo.	CS
327	CMPS101	A	BINF223	Intro. to bio.	Biology
835	BINF223	B	CMPS101	Algo.	CS
835	BINF223	B	BINF223	Intro. to bio.	Biology

Natural Join Example, cont'd

$R \bowtie S = \pi_{(attr(R) \cup attr(S))} (\sigma_{R.A1=S.A1 \text{ AND } R.A2 = S.A2 \text{ AND } \ldots \text{ AND } R.Ak=S.Ak} (R \times S))$

Step 2:

$\sigma_{R.A1=S.A1 \text{ AND } R.A2 = S.A2 \text{ AND } \ldots \text{ AND } R.Ak=S.Ak} (R \times S)$

Enrollment	Course

studentID	courseID	grade	courseID	description	department
112	CMPS101	C	CMPS101	Algo.	CS
327	CMPS101	A	CMPS101	Algo.	CS
835	BINF223	B	BINF223	Intro. to bio.	Biology

Natural Join Example, cont'd

$$R \bowtie S = \pi_{(attr(R) \cup attr(S))} (\sigma_{R.A1=S.A1 \text{ AND } R.A2 = S.A2 \text{ AND } \ldots \text{ AND } R.Ak=S.Ak} (R \times S))$$

Step 3:

$$\pi_{(attr(R) \cup attr(S))} (\sigma_{R.A1=S.A1 \cdots R.Ak=S.Ak} (R \times S))$$

Enrollment	Course

studentID	courseID	grade	description	department
112	CMPS101	C	Algo.	CS
327	CMPS101	A	Algo.	CS
835	BINF223	B	Intro. to bio.	Biology

Natural Join Example, cont'd

Enrollment(sid, cid, grade)

Course(cid, cname, instructor-name)

- Suppose you want: Course-grade(sid, cid, cname, grade)
- $\pi_{(sid, cid, cname, grade)}(\text{Enrollment} \bowtie \text{Course})$

- What happens when R and S have no common attributes?
- What happens when R and S have only common attributes?

Semi-Join: $R \ltimes S$

- Meaning: $R \ltimes S = \pi_{attr(R)} (R \bowtie S)$

1. Compute <u>Natural Join </u>of R and S
2. Output is the projection on <u>just the attributes of R</u>

- Find all courses that have some enrollment:
  Course $\ltimes$ Enrollment
- Find all faculty who are advising at least one student:
  Faculty $\ltimes$ Student

- How does Semi-Join relate to EXISTS in SQL?

Semi-Join Example:
Enrollment ⋉ Course

- **Start with Natural Join, then project on attributes of the first relation**

Enrollment	Course

studentID	courseID	grade	description	department
112	CMPS101	C	Algo.	CS
327	CMPS101	A	Algo.	CS
835	BINF223	B	Intro. to bio.	Biology

⋈

studentID	courseID	grade
112	CMPS101	C
327	CMPS101	A
835	BINF223	B

Enrollment ⋉ Course

Semi-Join Example:
Course ⋉ Enrollment

- Start with Natural Join, then project on attributes of the first relation

Course	Enrollment

courseID	description	department	studentID	grade
CMPS101	Algo.	CS	112	C
CMPS101	Algo.	CS	327	A
BINF223	Intro. to bio.	Biology	835	B

⋈

courseID	description	department
CMPS101	Algo.	CS
BINF223	Intro. to bio.	Biology

Course ⋉ Enrollment